Developing a Successful Metrics Program

Title: Developing a Successful Metrics Program
Presenters: Dr. Linda H. Rosenberg, Lawrence E. Hyatt
Track: Metrics
Day: Wednesday, April 24, 2995
Keywords: Metrics, software, quality, risk

Abstract:

This paper discusses how affordable metric programs can be applied to help project managers and developers evaluate the quality of their project's products and to help them evaluate and track project risks. A core set of relevant attributes and metrics are developed for all software development phases. The Goal/Question/Metric paradigm (GQM) is used to demonstrate how a meaningful metrics program can be started and uses data from projects at Goddard Space Flight Center (GSFC) to demonstrate some analysis and application techniques.

This paper supplies both project managers and software developers with techniques to initiate a metrics program that yields timely, relevant, usable information at minimal cost.

1. Introduction

At the Goddard Space Flight Center (GSFC), and indeed across NASA, there is a focus on producing "quality software". This focus is not just within NASA; it can be seen in almost any organization where software results contribute to success, from programmable microwaves to watches to toys - quality products depend on quality software. Everyone agrees that quality is important, but few agree on what quality is or how to measure it. Kitchenham notes that 'quality is hard to define, impossible to measure, easy to recognize'.[8]

The Software Assurance Technology Center (SATC) was established in 1992 in the Systems Reliability and Safety Office at NASA's Goddard Space Flight Center (GSFC). The SATC was founded as a center of excellence in software assurance, dedicated to making measurable improvement in the quality and reliability of software developed for GSFC and NASA. The SATC has programs in four areas: Software Standards and Guidebooks; Software Metrics Research and Development; Assurance Tools and Techniques; and Project Support and Outreach.

The objectives of the software metrics program of the SATC encompass three areas: Quality Assessment; Risk Management and Control; and Process Improvement. To accomplish these objectives, the metrics program works with project managers and developers (both NASA and contractor) to collect measurements and develop metrics that assist them in evaluating the quality of the products (requirement documents, design

documents, code, test plans, etc.) and the risks to their projects. This work resulted in the formalization of the Software Quality Metrics Program.

But developing a metrics program is not easy. It has many possible pit-falls that can lead to the ruin of the metrics program itself and possibly the project if incorrectly applied. Section 2 discusses the foundation of a software quality metrics program based on the evaluation of quality and risk projections. Practical components of a metrics program such as costs, benefits and the development stages are discussed in Section 3. In Section 4, a basic metrics program is outlined with core attributes and metrics that are applicable to any software development project. How to develop and implement a metrics program is discussed in Section 5. The Goal/Question/Metric Paradigm is explained and then applied to a sample metrics program. This example starts by defining goals, developing applicable questions, and demonstrating the metric analysis using GSFC project data. The paper concludes with data collection and screen development guidelines and a discussion of a tool developed at GSFC for capturing, maintaining and distributing data on-line using the World Wide Web.

2. Software Quality Metrics Program

The SATC has developed a Software Quality Metrics Program and associated metrics that supports risk management and quality assessment of the processes and products of software development projects. The SATC program, shown in Figure 1, meets several criteria that are important to NASA software project managers. It contains dynamic elements, so that projections in time can be made to effect the direction of the project. It takes into consideration project goals and milestones, so that it can be tailored for the particular project. It assists in determining risks, so that management action can be taken. The combination of these three elements makes the SATC program unique.

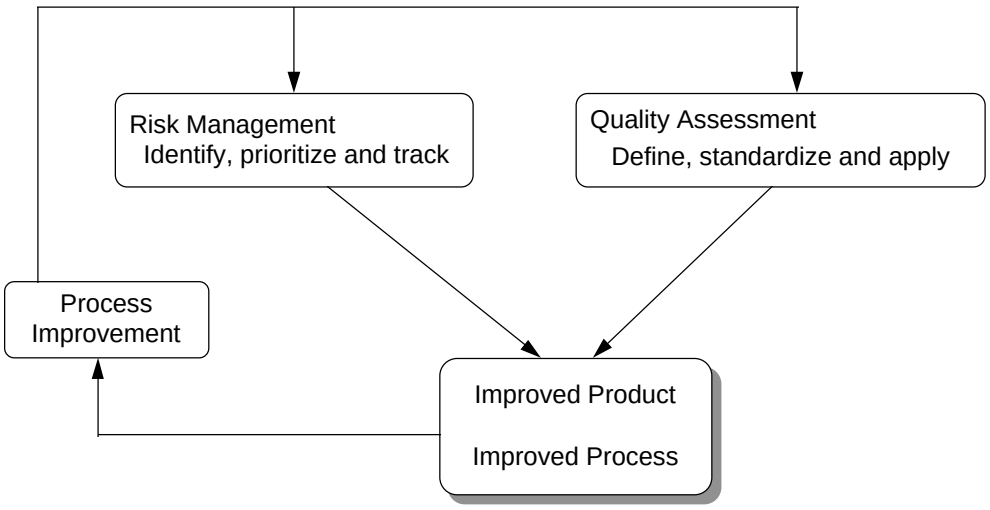


Figure 1: SATC Software Quality Metrics Program

In the top portion of Figure 1, the two primary objectives are stated. First, risk management - the relevant project risks must be identified, prioritized, then tracked through the life of the project. By identifying and prioritizing risks, project management will customize the metrics program to specific areas relevant to their project. In assessing the quality, relevant product quality attributes must be defined for the project. The attributes chosen customize the metrics program to the project goals. Corporate and industry standards must be identified then applied.[1]

Applying these objectives leads to two actions: improve the product and improve the process. Although many software development organizations espouse that a well defined process results in a good product, there is little evidence that conformance to process standards guarantees good products. [7] Therefore, metric programs should be comprehensive - dynamically evaluate the development process as well as statically evaluate the products. Many organizations, such as the Software Engineering Institute at Carnegie Mellon University, and the Software Engineering Laboratory at Goddard Space Flight Center have developed very comprehensive programs for software development that focus on process improvement. The metrics programs developed by these institutes support process improvement with little emphasis on product evaluation. The SATC metrics program discussed in this paper focuses on the evaluation of life cycle products using static metric analysis and is complementary to these programs. Project developers and managers need both sets of metrics.

[1] Reference "A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality" by L. Hyatt in the "Quality" track of this conference.

3. Metric Program Practical Components

3.1 Cost

It is difficult to pin down the costs of a metrics program because metrics are usually just one aspect of an overall improvement program. When investigating the feasibility of starting a metrics program, it is often found that managers are individually collecting some form of data. This decreases initial program start up cost. Accurate and complete measurements are not inexpensive; comprehensive metrics programs for software products and process annual costs can be 2 to 3 percent of the total software budget for collecting hard data.[6] Attempts to pin down the cost of metrics hide the real issue, however, developers don't really have a choice. The cost of not implementing a software metrics program can be measured in terms of project and business failures. Those projects and companies who make the investment in metrics have a competitive advantage over those who do not. They have the advantage of more informed and timely decisions that will ultimately make them more successful, with the best track records in terms of bringing software projects to completion and achieving high levels of user satisfaction.[5]

3.2 Benefits

Table 1 summarizes some of the costs versus benefits of metrics programs. This is not a comprehensive list but highlights the more visible aspects. The largest benefit as noted in the section above is the increase in knowledge of product and project status and the ability to assess risks and facilitate contingency planning.

ITEM	COST	BENEFIT
Training	Class costs Less time on project	Future expertise Consistency, standardization
Management	More management time	Less management time
Engineering	Start up costs Data collection time Analysis time	Increase productivity Reduce development time Fewer defects Reduce defect find/fix time Reduce maintenance Increase reusability Increase user satisfaction
Capital Expenses	Purchase hardware/software	

TABLE 1: Cost versus Benefits [6]

It is difficult, if not impossible, to place a dollar amount on the benefits of a metrics program because as in the case of risk management, you are trying to measure something that did not happen. The benefits derived are also not only applicable to the current project but to future projects. As with any new project, whether it is implementing a new engineering design or a metrics program, start up costs are high. But as management and staff become familiar with the tasks and tools are developed, the costs decrease to a low maintenance level.

3.3 Phases of Metric Programs

As products or companies initiate metric programs, either voluntarily or because of contractual requirements, personnel progress through 4 stages of acceptance. Initially, there is resistance to the idea of a metrics program, refusal to interact and feelings of being threatened or possible mistreatment resulting from the metrics. The metrics team need very persuasive and persistent social skills to initiate the data collection. This is followed by a reaction stage; the metrics program and data collection are not going to dissolve and the development team must learn to live with it. The metrics team now has the developers attention and needs to show relevant value added by the metrics program. The third stage is acceptance of the metrics program; the development team recognizes the benefits and incorporates the metrics into the development infrastructure. The final stage is the dependence of the development team on the metrics in the decision making process. These stages are shown in Figure 2.

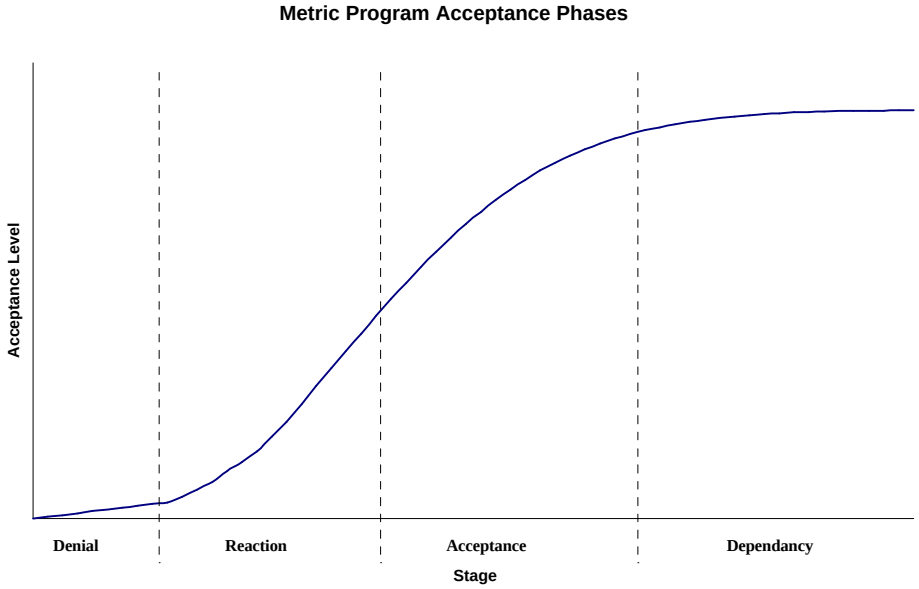


Figure 2: Metric Program Stages

The length of time for each phase is generally project dependent but influenced by many factors. The first projects to adopt a metrics program will have longer initial phases, the entire concept of software development measurement is new and approached with trepidation. As a project progresses through the stages, the costs of the metrics program decreases and the benefits derived increase.

4. Basic Metrics Program

Metric programs are perceived to be expensive and hence unaffordable by software development projects as discussed in the previous sections. But what is often not recognized by development organizations, is that each project does not need a totally unique metrics program, nor does each project need to incorporate all possible metrics to have a successful metrics program. While it is true that every project is different, there are also many aspects of all projects that are similar. All projects have a mission to complete. All projects have a set of requirements that must be satisfied. Some design format must be used to move the project from the requirements to code to ensure all requirements are fulfilled and to ensure the components of the code work as a cohesive whole. From the design, code is written. Standards and guidelines for module formats exist for most high level programming languages. The testing phase has two primary objectives, validating the software meets the specifications of the requirements and verifying the correctness (error free). These common traits found in all software development projects allow for different projects to use a similar metrics program, with a common set of attibutes representing quality and risk characteristics.

The purpose of core attributes is to provide project managers with a common set of characteristics to assess project quality and risks, regardless of the application environment or the implementation details. The core metrics provide a starting point for the project, decreasing initial metric program development costs and providing general interpretation guidelines. From the core set, additional attributes and metrics can be defined to supply information that is project specific or addresses a specific area of risk that is of concern to the project.

4.1 Core Attributes

Prior to choosing the metrics for each life cycle phase, the measurable attributes must be defined. These attributes are features or characteristics that affect the quality of the product and can be used to estimate future risks. They are applicable to any project. Projects can then place emphasis on the attributes that are important to their specific goals.

The attributes used by the SATC are defines as follows:

Ambiguity - multiple interpretations
Completeness - all components contained within
Comprehensiveness - single test specification per requirement/design feature

Consistency - agreement of all levels with higher level documents
Correctness - specifications are fulfilled
Documentation - description of the content
Efficiency - availability and usage of resources
Error detection - rate errors located and repaired
Feasibility - ability to complete with specified resources
Maintainability - ease to locate and correct faults
Reuse - ability to apply or use module in a different content
Schedule - definitions of milestones and their attainability
Structure/Architecture - structure or framework of the module
Testability - rate of testing and code correction
Volatility - intensity and distribution of changes
Verifiability - ability to trace each component from requirement to software to test

4.2 Core Metrics

Core metrics provide a means of quantifying the attributes. Table 2 associates the attributes and core metrics for each software development phase and serves as a basic foundation that can be supplemented for project specific goals. There is not a one-to-one mapping between the attributes and the metrics. In many cases, the metrics are applicable to more than one attribute, and the attributes are measured through a combination of metrics.

Life Cycle Phase	Attributes	Core Metrics
Requirements	Ambiguity Completeness Consistency Volatility Verifiability	Baseline counts Traceability Terminology Structure
Design	Ambiguity Completeness Consistency Volatility Verifiability	Trace matrices Complexity
Implementation	Comprehensiveness Correctness Documentation Feasibility Structure/Architecture Reuse Maintainability	Errors/faults/changes Module size and complexity Documentation Traceability Resources
Testing	Testability Error Detection	Errors/faults/changes Coverage - Trace matrices

TABLE 2: Life Cycle Phase Attributes and Core Metrics

The core metrics in Table 2 are defined as follows:

Baseline counts - initial or base number of requirements
Complexity (of design) - data flow within and between segments
Documentation - internal or external commenting
Errors/faults/changes - count, type, criticality, and time to find/fix
Module complexity - logic, data, and calling within a module
Module size - line or token counts within a module
Resources - personnel hours or effort expended
Structure - level/depth within document requirement is specified
Terminology - phraseology of requirements e.g., use of imperatives, continuances, weak phrases
Traceability - requirements traced to design component to code module to test number

5. Developing and Implementing a Metrics Program

5.1 Where to Start

Once a developer decides to implement a metrics program, the next step is *How*. How a metrics program is developed can control its success or failure. One approach is to investigate tools available for metrics collection, purchase the tool, then collect and attempt to apply whatever metrics are provided by the tool. This may work but has a major hurdle - what will the data collected tell the management and developers about their specific project; how will it help in evaluating the quality and projecting project specific risks. Data collected just because it is available has minimal value at best and usually ends up a waste of resources.

Successful metrics programs generally begin by focusing on a problem. At the start of the metrics program, risk and quality goals must be established that address the problem as discussed in Section 2. Related questions that management wants answered are identified then the data that is needed to answer these questions are specified. This leads to the tool specification for purchase or in-house development. Data collection can be expensive if not carefully monitored - the temptation is high to collect all possible data and decide how to use it later. This type of process generally leads to failure because the quantity of data now overshadows the quality. Extraneous data should be discarded.

The sooner benefits are seen by both management and developers, the faster metrics programs progress through the acceptance stages in Figure 2. Metric programs should be designed to show visible benefits as soon as possible, this is the key to continued support.

5.2 How to Start

5.2.1 Goal/Question/Metric (GQM) Paradigm

The Goal/Question/Metric (GQM) Paradigm is a mechanism that provides a framework for developing a metrics program. It was developed by Vic Basili and H. Dieter Rombach at the University of Maryland as a mechanism for formalizing the characterization, planning, construction, analysis, learning and feedback tasks. The GQM paradigm was developed for all types of studies, in particular studies concerned with improvement issues. The paradigm does not provide specific goals but rather a framework for stating goals and refining them into questions to provide a specification for the data needed to help answer the goals.[1,2]

The GQM paradigm consists of three steps:

1. Generate a set of **goals** based upon the needs of the organization.
2. Derive a set of **questions**.

3. Develop a set of **metrics** which provide the information needed to answer the questions.

1 - Generate a set of goals based upon the needs of the organization - Determine what it is you want to improve. This provides a framework for determining whether or not you have accomplished what you set out to do. Goals are defined in terms of purpose, perspective and environment using generic templates:

- *Purpose:* To (characterize, evaluate, predict, motivate, etc.) the (process, product, model, metric, etc.) in order to (understand, assess, manage, engineer, learn, improve, etc.) it.
- *Perspective:* Examine the (cost, effectiveness, correctness, defects, changes, product metrics, reliability, etc.) from the point of view of the (developer, manager, customer, corporate perspective, etc.)
- *Environment:* The environment consists of the following: process factors, people factors, problem factors, methods, tools, constraints, etc.

2 - Derive a set of questions - The purpose of the questions is to quantify the goals as completely as possible. This requires the interpretation of fuzzy terms within the context of the development environment. Questions are classified as product-related or process-related and provide feedback from the quality perspective. Product-related questions define the product and the evaluation of the product with respect to a particular quality (e.g., reliability, user satisfaction). Process-related questions include the quality of use, domain of use, effort of use, effect of use and feedback from use.

3 - Develop a set of metrics and distributions that provide the information needed to answer the questions - In this step, the actual data needed to answer the questions are identified and associated with each of the questions. As data items are identified, it must be understood how valid the data item will be with respect to accuracy and how well it captures the specific question. The metrics should be objective and subjective and should have interpretation guidelines, i.e., what value of the metric specifies the product higher quality. Generally, a single metric will not answer a question, but a combination of metrics is needed.

Once goals are defined, questions derived, and metrics developed, matrices are created to indicate their relationships. The first matrix is from goal to question, the second from question to metric. These allow the developers to identify metrics applicable to multiple questions and to guarantee for each goal there is more than one question and more than one metric. A summary matrix from goal to metric can also be developed. The matrices identify single relationships that may not be cost effective.

5.2.2 Goal/Question/Metric (GQM) Paradigm Example

The most effective way to understand a methodology is to review an example. This section demonstrates how a small metrics program would be developed using the

GQM. The program starts with the goals, questions and proposed metrics, then demonstrates how GSFC data could be used to answer the questions and satisfy the goals.

Four sample goals using the templates are shown in Figure 3. Goals 1 & 4 follow the purpose template; Goal 3 uses the perspective template; and Goal 2 evaluates the environment. The goals are not specific and are not limited to evaluating a specific phase within the development life cycle. The goals are general and could be adapted with minor modifications to any project development.

G1: To predict the schedule in order to manage it.

G2: The system must release on time with at least 90% of the errors located and removed.

G3: Examine the maintainability risk from the point of view of the customer.

G4: To evaluate the product in order to improve it.

Figure 3: GQM Goals

Now questions are derived to quantify the goals. In this example only a limited number of questions are specified. A question often supports more than one goal. It is listed under the primary goal and secondary goals shown in parenthesis ( ). The metrics needed to provide the answers to the questions is then chosen and is shown in italics. At this point, COTS (Commercial off the shelf) tools are investigated, data collection procedures are developed, and forms are designed. (These concepts are discussed in a later section.) The goals with questions and metrics are shown in Figure 4.

G1: To predict the schedule in order to manage it.
Q1: What is the actual vs. expected effort level? (G2) *effort*
Q2: How stable are the requirements? (G4) *req. cnt & modification*

G2: The system must release on time with at least 90% of the errors located and removed.
Q3: What methods are used for testing? (G2) *% tests/method*
Q4: When will 90% of the errors be found and all priority 1 errors closed? (G1, G4) *errors, effort, size*
Q5: What is the discrepancy rate of closure? (G1) *errors, closure status*

G3: Examine the maintainability risk from the point of view of the customer.
Q6: What percentages of modules exceed the guidelines for complexity and size? (G2, G4) *complexity, size*

G4: To evaluate the product in order to improve it.
Q7: What modules are "high risk"? (G2, G3) *complexity, size, errors*

Figure 4: Goals/Questions/Metrics

Matrices similar to those in Figures 5 and 6 are completed. First the goals are related to the questions in Figure 5 showing direct and indirect correlations.

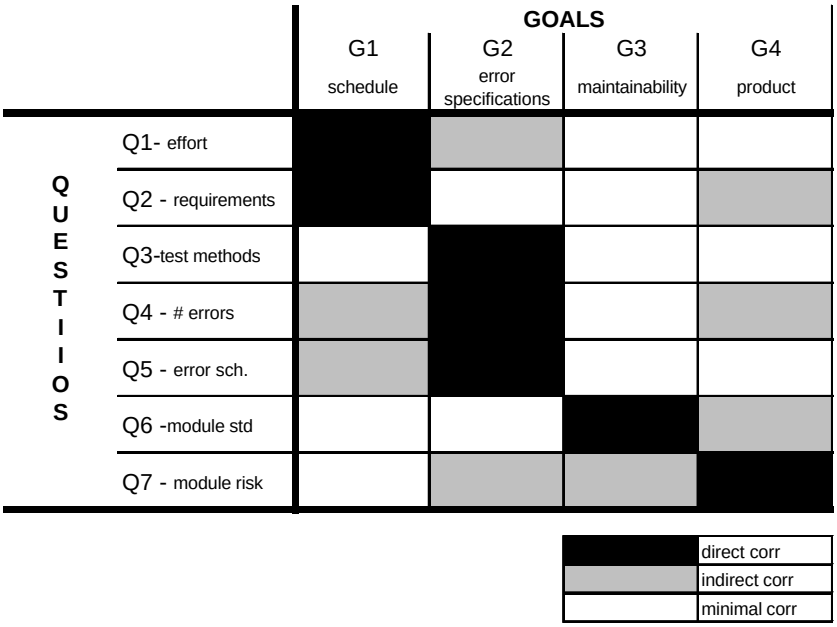


Figure 5: Goals --> Questions

From Figure 5 it can be determined that Goals 1, 2 and 4 have multiple questions supply data but Goal 3 is weak. Based on the priority of the project, more questions, and hence data, might be needed to support this goal. This matrix also indicates that except for Question 3, most questions support multiple goals. Again, the project must determine the acceptability of this limited support based on their objectives.

Figure 6 is the matrix that relates the questions to the metrics. A metric that supports only one question may not be cost effective to collect.

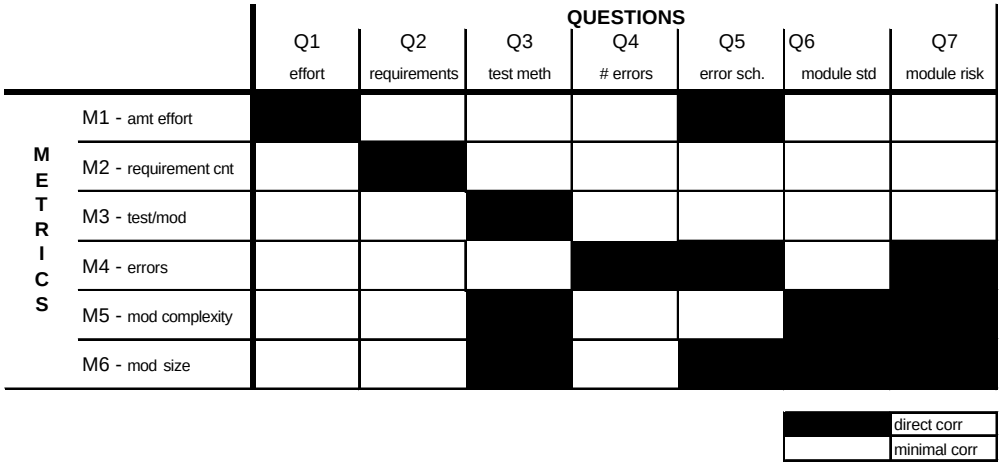


Figure 6: Questions --> Metrics

5.2.3 Goal/Question/Metric Data Analysis Example

In this section the goals and supporting questions are discussed using data from GSFC projects. It is SATC policy that all reported data is anonymous so project characteristics such as purpose (ground or flight), application (command and control, simulator, data), and language (FORTRAN, C or C++) are used as identification for comparisons across projects.

After the metrics are presented and analyzed for each Goal/Question section, a status of the risk will be discussed. The SATC model applies the following classifications for risk levels:

LOW - Very likely to meet objectives if current trend continues. Does not need contingency plans.
MODERATE - Based on current trend, likely to meet objectives. Should have contingency plans.
HIGH - Not likely to meet objectives based on current trend. Implement contingency plans immediately.

G1: To predict the schedule in order to manage it.
Q1: What is the actual vs. expected effort level? (G2) *effort*
Q2: How stable are the requirements? (G4) *req. cnt & modification*

The metrics for this goal and questions are effort, requirement and modification counts. Effort data refers to the number of hours personnel work on a specific activity, such as training, requirements, design, code and unit test, system test or management. The SATC masks this data using an anonymous personnel identification number to prevent management from using it as an personnel evaluation tool. Management is given a weekly report showing total hours per activity. Requirement and modification counts are a straight count of the number of requirements that the project must fulfill and a count of the number of increases or decreases to the base number of requirements.

Prior to the start of a project, management should estimate the amount of time required for each phase. This allows for management to have the correct development skill mix available at the appropriate time. Figure 7 shows the percentage of time four different development applications spent in the phases. The top two are divisions at GSFC; SPA tests new software engineering development concepts using extensive prototyping, and SEL is well established in developing Flight dynamic software. The bottom two graphs are from R. Grady's book *Practical Software Metrics for Project Management and Process Improvement*. [5]

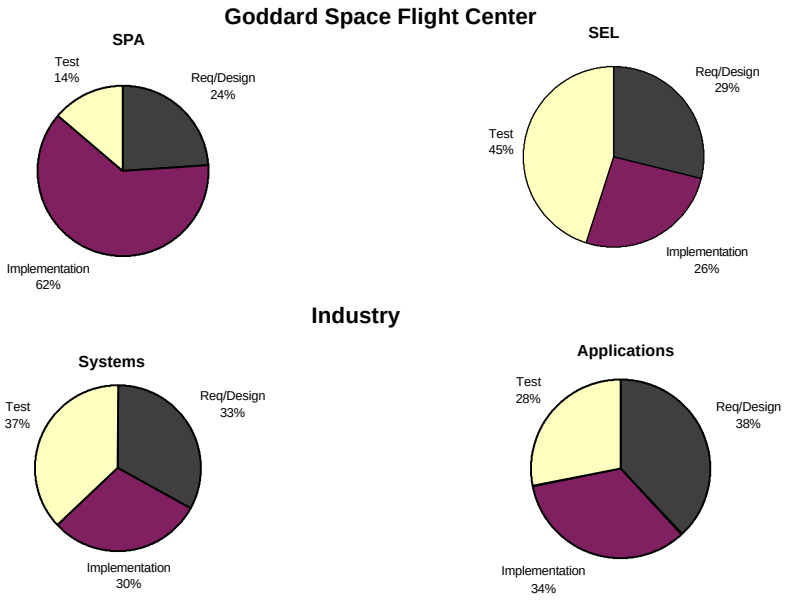


Figure 7: Time per Phase by Application

Figure 8 is the Rayleigh Manpower Curve for effort expenditure for a typical software project. [11] Projects ramp up to full speed fairly quickly, then taper off as the maintenance phase approaches. Applying this curve assists managers from having too few or too many personnel.

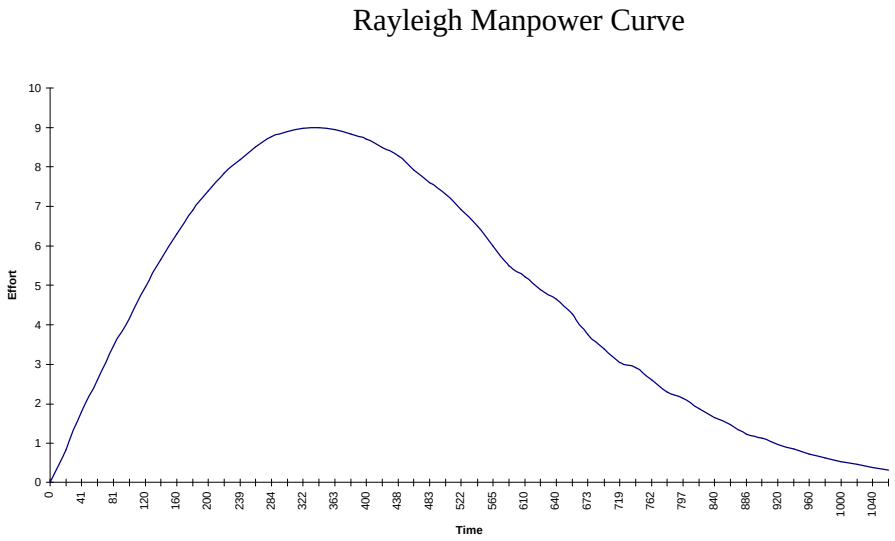


Figure 8: Effort Expenditure Curve

Figures 9 & 10 address Question 2, the stability of the requirements and the modifications to requirements. Industry has shown that unstable requirements increase the risks to final project. Developers cannot design or code a moving target and changes later in the life cycle have ripple effects that may impact both product and schedule risk. Figure 9 indicates the number of requirements is stabilizing and looks good but Figure 10 indicates there are still numerous changes.

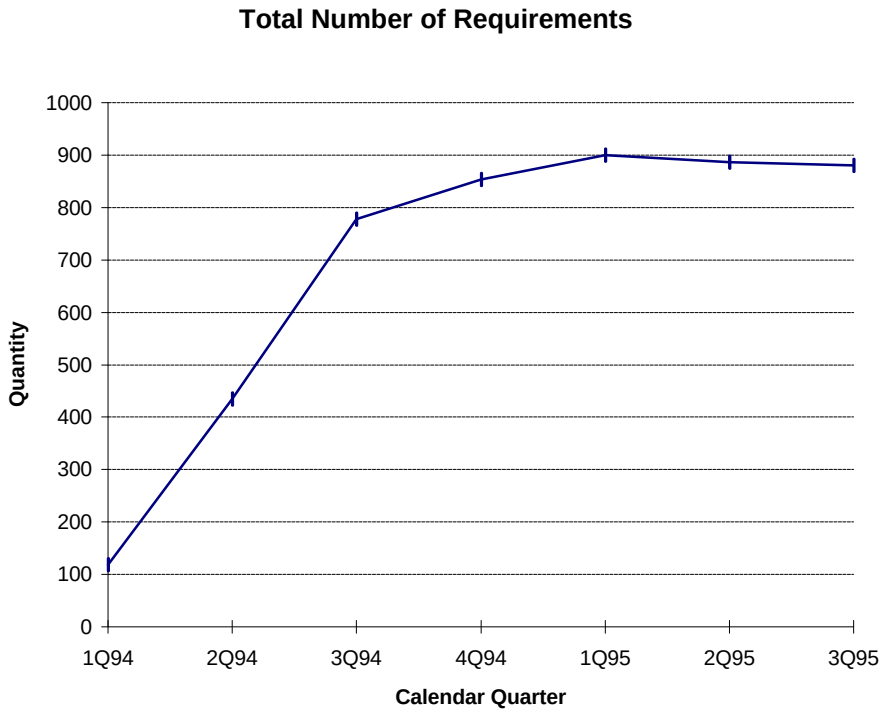


Figure 9: Requirement Count - Stability

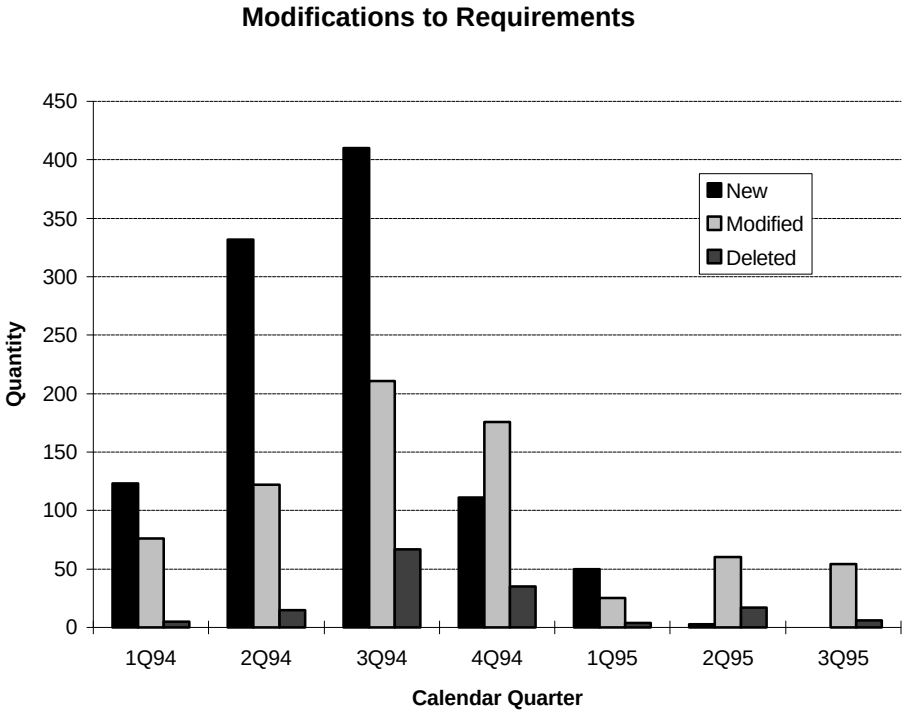


Figure 10: Requirement Modification - Stability

Project managers apply the two effort concepts, percentage of effort per activity and total effort, to their projects by collecting data and comparing it to expected results shown in Figures 7 and 8. Extensive deviation would indicate a potential problem area and risk to the schedule. Schedule slippage can result in decreased test time and hence further risk.

The requirement data indicates the schedule is still at risk because the requirements are still being modified, but since the count is stabilizing, the added risk is minimal.

G2: The system must release on time with at least 90% of the errors located and removed.
Q3: What methods are used for testing? (G2) *% tests/method*
Q4: When will 90% of the errors be found and all priority 1 errors closed? (G1, G4) *errors, effort, size*
Q5: What is the discrepancy rate of closure? (G1) *errors, closure status*

Figure 11 specifies the testing methods that will be applied for this project. All requirements have been assigned a testing method, 2% of the requirements will be verified by demonstration and 6% by inspection. It is up to project management to determine which requirements are verified by each method and if the verification methods are acceptable.

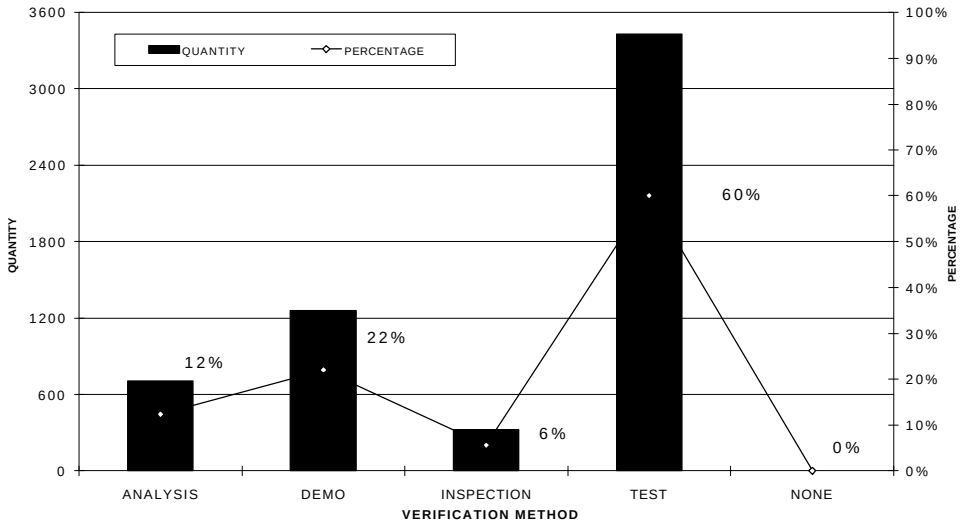


Figure 11: Testing Methods

Question 4 specifies that 90% of the errors be located. This implies the ability to estimate the total number of errors in the software. One industry guideline that is also recommended by the SEL is to expect approximately 7 errors per 1000 Source Lines of

Code. This estimate is helpful in an overall estimate of the number of errors, but does not take into account the rate at which errors are found. The SATC is working to release the Waterman Error Trending Model for determining the status of testing by projection of the number of errors remaining in the software and the expected time to find some percentage of errors. This model uses the Musa Reliability model and the Raleigh curve for effort estimations. The SATC is developing this model rather than using the standard Musa model because it is less sensitive to data inaccuracy and provides for non-constant testing resource levels. Figure 12 is an example of the model's application.

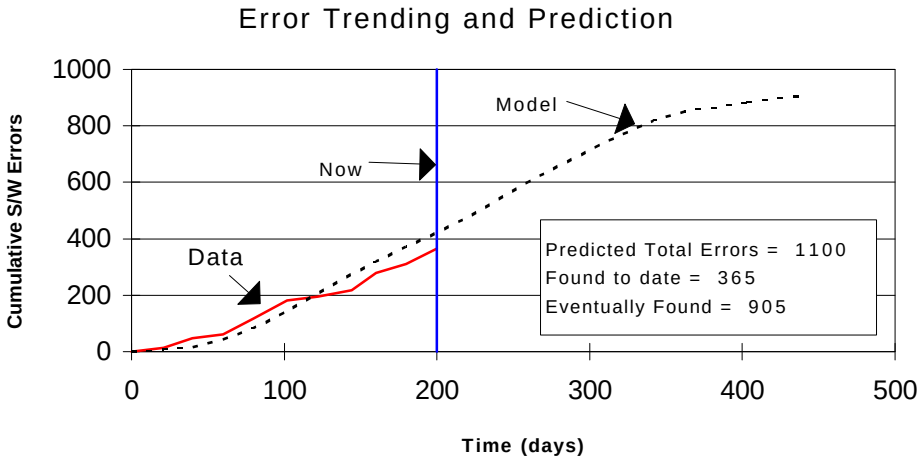


Figure 12: Waterman Error Trending Model

If it is assumed that this project should release its product shortly, Figure 12 indicates that only about 40% of the errors will be found.

Figure 13 indicates the rate at which discrepancy reports are closed. No discrepancies have been reported since 3/20/95 and all were assigned a status by 5/8/95. Trend analysis indicates all discrepancies will be closed by the conclusion of this phase.

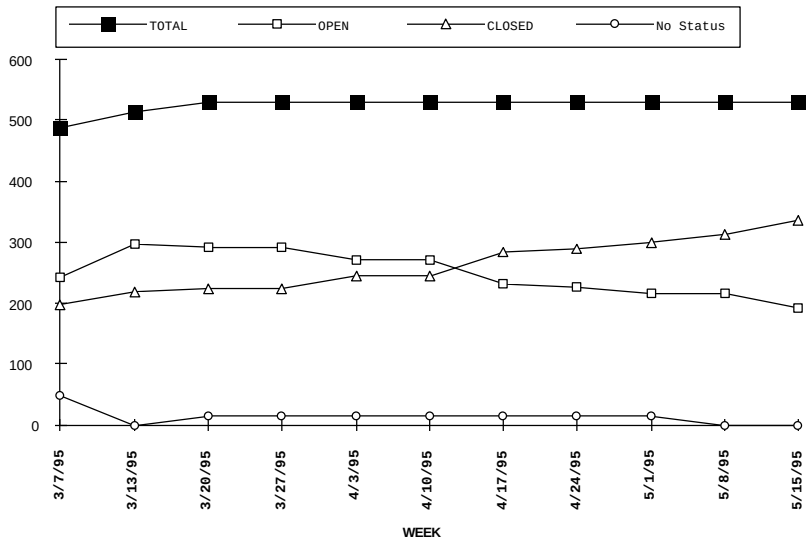


Figure 13: Discrepancy Report Closure Rate

Since Question 4 indicates Priority 1 discrepancies closure is critical, Figure 14 assesses the status of open/closed by priority and indicates that all Priority 1 should close in one more week (8/28/95). This graph also indicates an extraneous problem with database maintenance - Priority 3 discrepancies are closed before they are opened.

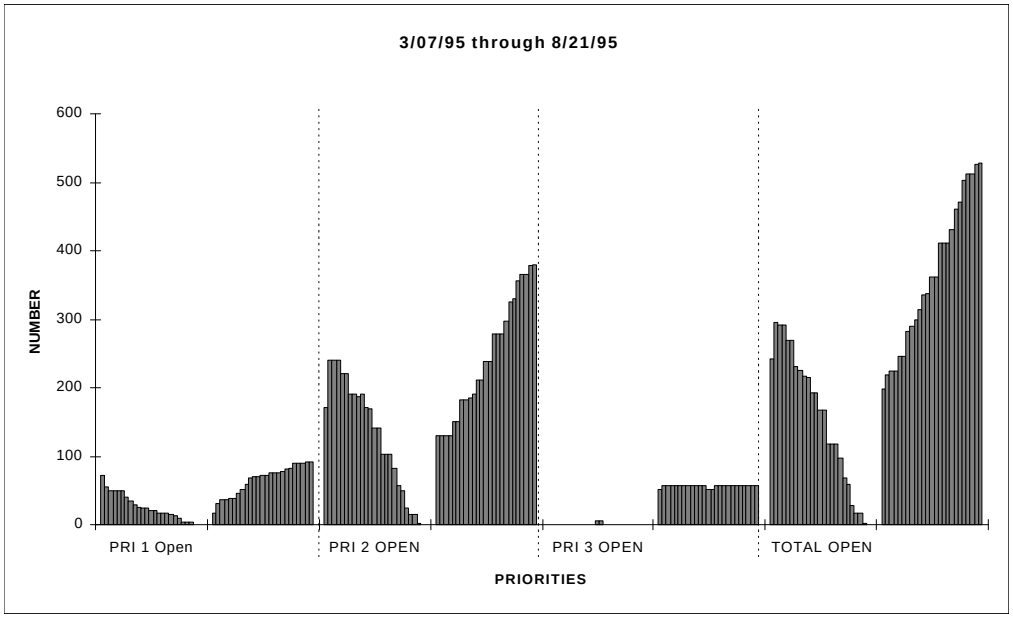


Figure 14: Discrepancy Closure Rate by Priority

The analysis of the data for this section indicates that all components have been assigned a verification method, discrepancy reports are expected to be closed within the specified time frame, including Priority 1, which were a low percentage of the total number. The risk of not meeting this goal of releasing on time is negligible. However, using the results in Figure 12, the project will not meet the specified 90% errors found in Question 4. This increased the risk to Goal 2 to high.

G3: Examine the maintainability risk from the point of view of the customer.
Q6: What percentage of modules exceeds the guidelines for complexity and size?
(G2, G4) *complexity, size*

Maintaining code encompasses locating and fixing an error or altering existing code in response to a change request. Both cases require the ability to locate sections of code where a specific task is done or tracking values through code. Modules of code that are smaller in size, lower in complexity and have a high comment percentage are generally easier to change. The amount and type of variables transferred between modules (fan in/fan out) also affects the maintainability. Most questions need a combination of metrics to fully support the goal. For this question we analyze a size/complexity correlation for an initial answer.

Figure 15 is a graph template developed by the SATC to use as an indicator of the module risk levels. The x-axis represents the number of executable statements in a module; the y-axis is the extended cyclomatic complexity (number of test paths) for the module. There are many different guidelines for both measures as to when risk increases or the code is decreases in acceptability. The parameters in Figure 15 are based on guidelines from various industry, military and NASA sources as well as error correlations from GSFC data.

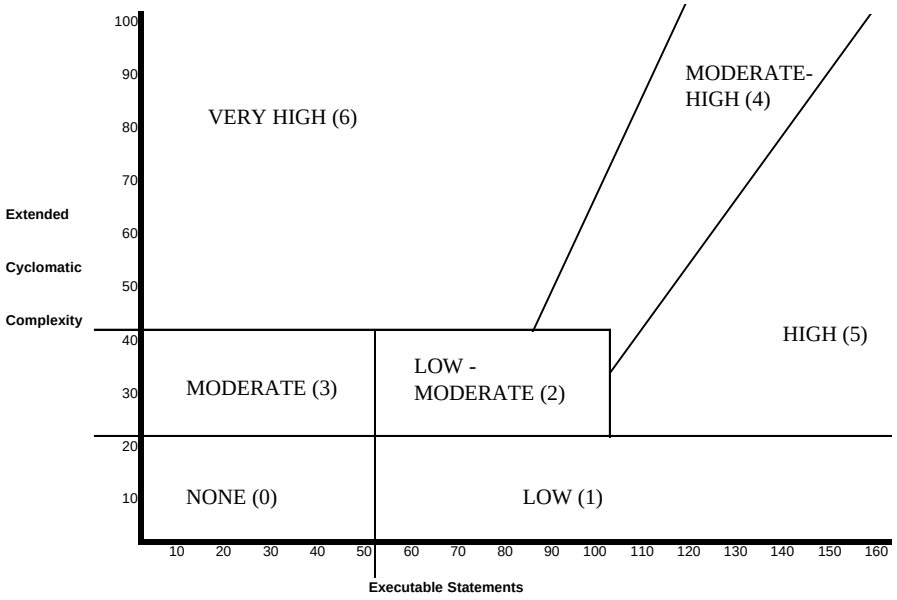


Figure 15: Classification of Module Risk

To apply the graph in Figure 15, each module of code (procedure/function/class/method - language dependent) is plotted as shown in Figure 16. The percentage of modules in each region and a list of module names in Regions 4, 5, and 6 are supplied to the project management. It is recommended that developers further investigate the modules in these regions using further using metrics such as fan in/ fan out, comment percentage and number of errors. One observation made by the SATC is that C and C++ code have a lower percentage of modules in Regions 4, 5 and 6 than FORTRAN.

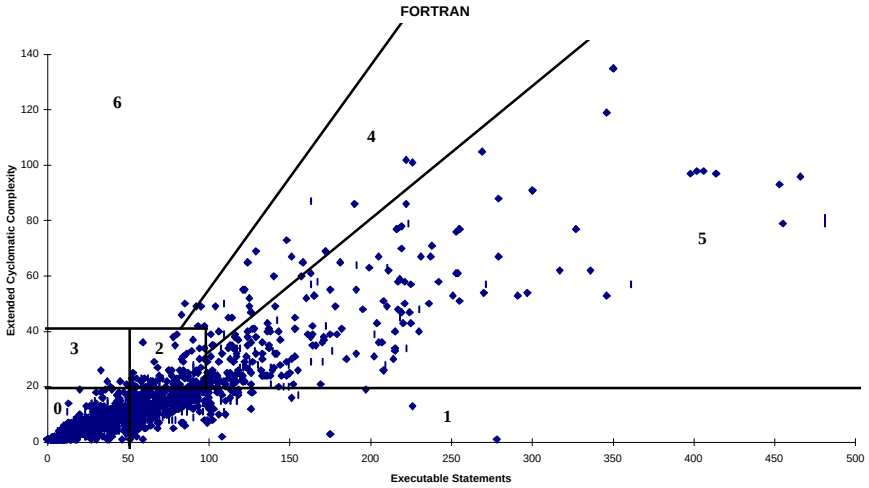


Figure 16: Modules at Risk in a FORTRAN Project

For the project code shown in Figure 16, there is 22% of modules in Region 5 where the risk is High, and 9% of the modules in Region 4 (Moderate to High risk). Additional metrics such as documentation, structure and data flow should be investigated for the modules. Based on current information, this project would be rated at a moderate risk of meeting Goal 3 - maintainability.

G4: To evaluate the product in order to improve it.
Q7: What modules are "high risk"? (G2, G3) *complexity, size, errors*

To answer this question, multiple metric inputs are needed. One factor is the risk from Table 3. The number of errors by criticality also serves as an input. Comment percentage also supplies information relative to risk. Factors such as fan-in/fan-out and internal data flow also influences risk but are not shown in Table 3.

Module ID	Size/Complexity Risk	Errors (crit 1 2 3)	Comment %	Risk Total
113	none	12 (0 0 12)	40%	low
76	none	10 (0 1 9)	37%	low
158	mod-high	10 (0 2 8)	37%	moderate
57	high	9 (1 2 6)	25%	high
2	very high	8 (1 2 5)	27%	high
95	moderate	7 (1 2 4)	15%	moderate
152	very high	1 (1 0 0)	3%	high

Table 3: Module Risk Value

For this goal, the objective is to identify areas that management should focus on to improve. Table 3 indicates the modules 158, 57, 2 ,95 and 152 all need further investigation and possibly reengineering. Since these 5 modules are less than 1% of the total modules, the risk of the code is negligible.

5.2.4 Goal/Question/Metric Example Conclusion

In the previous sections the risk levels for the goals was estimated based on the metrics analysis. In presentations to management, especially upper levels, a version of the "fever chart" has been found to be effective. However, since the colors used in fever charts (green, yellow, red) do not reproduce well, symbols are used. These symbols follow user interface guidelines using the darkest shade for the most important concept and a relevant symbol ($) to reinforce the meaning. Figure 17 summarizes the risks for the goals in the GQM example.

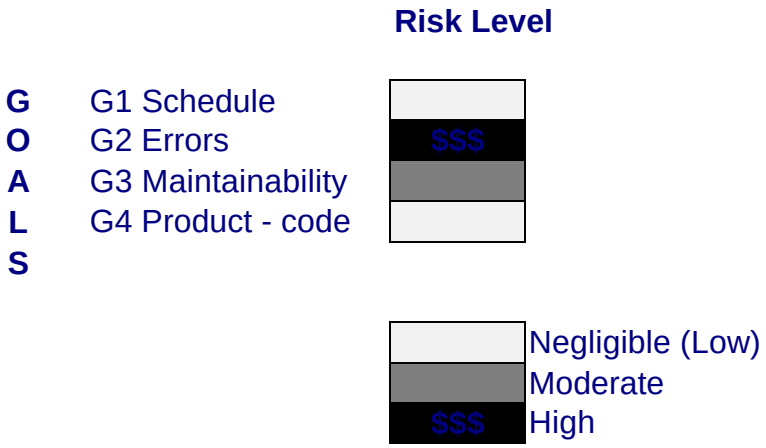


Figure 17: Summary of Goal Risk Level

6. Data Collection

Although the focus of the paper is on relevant metric analysis, some discussion is needed on general data collection guidelines - what to collect and how to collect it. If data input is invalid, the resulting analysis will be skewed.

6.1 Data

Metric programs are based on raw data and measurements collected from multiple sources. Successful metrics programs should initially focus on what data should be collected and the format of the data collection. There is an overwhelming amount of data available that can be very expensive to collect and may not provide the answers. The SATC has developed four general guidelines for metric programs.

1. Low resource usage, non-intrusive

Although there is a cost for a metric program, it should not be so expensive that metric collection is not feasible. This "cost" includes both financial and personnel resources. Many organizations assume that the cost of measurement is so excessive that they cannot justify establishing a metrics program. While measurement is not free, it can be tailored to fit the goals and budgets of any software organization. A metrics program must be undertaken with the expectation that the return will be worth the investment. The costs must be incorporated in the project budget or there will be frustrations, attempts at shortcuts, and a failed metrics program.

Data collection for a metrics program should not be disruptive to personnel. Data collection should be an intricate part of the work structure, automated whenever possible. Automation increases the accuracy and decreases the interruptions. Automated input is also preferable, research indicates that electronic forms tend to be completed faster and more accurately than paper forms. Existing systems and data should be used whenever possible.

2. Non-threatening, assess projects and products, not people

The second guideline for a successful metrics program is to focus on measuring the project and the products, not the people. Metric programs can successfully evaluate processes, but the temptation to use the resulting metrics to evaluate the people performing the processes must not be succumb to or the metrics program has a short life span and is doomed to failure. When people feel threatened by metrics, they either do not produce the metrics or skew them to reflect positive results. Metrics programs used to evaluate personnel generally yield invalid results and have a very limited life span, hence wasting time and money.

Two concepts reduce the threatening influence of metric programs. One method is to keep all data anonymous, using identification numbers instead of personnel names. Using an independent third party to assign and maintain the id numbers helps insure confidentiality. The second concept that assists in reducing metric stress is education. Anyone associated with a metrics program should know the specific purposes and objectives of the metric programs, what metrics are being collected, and how they are collected (especially if the process is automated or invisible to the person). As the analysis is done, sharing the results and the conclusions not only reduces some of the metric induced stress, but allows personnel to input an alternative conclusion or an explanation for why metrics imply a specific conclusion.

3. Yield relevant, reliable information

When developing a metric program, it is tempting to collect as much data as possible since the data might be needed later, This approach usually leads to metrics programs that focus on where to store the data rather than how to analyze the data and use the results. The data collected should focus directly on the objectives of the project manager. All data that is collected should be used to provide immediate feedback to managers, not just stored for future use. Projects should collect what they can use and use what is collected. This often means starting with a small, limited set of data until that data is evaluated, the application understood and then the results applied. The key to

continued funding of metric programs is often showing management immediate visible benefits derived from the metrics; the promise of only future benefits often leads to reduced funding.

4. Adaptable across projects, language and application independent

Although a metrics program should focus on current objectives, it should be designed so it could be adapted for projects in other languages and in other application fields. Data should be sufficiently general to allow for comparisons to other projects, thus increasing the applicability of the metrics beyond a single projects and permitting general guidelines to be developed. Similar data can be stored in a historical database to provide information for continuous improvement and generic understanding of project development.

6.2 System Input

The quality of the system output is determined by the quality of the system input. Having an effective, usable metrics program depends on the accuracy and reliability of the data input by the users and by the system. Input from system users requires careful consideration to form or screen design. Some guidelines for form design are:

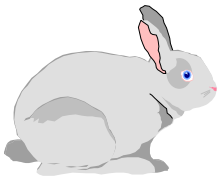
1. Effectiveness - Ensure that forms meet the purpose for which they are designed. Keep information relevant to task and avoid duplication of information.
2. Accuracy - Design forms to assure accurate completion - Use internal checks for sums (horizontal/vertical).
3. Ease of use/simplicity - Make forms easy to complete. Forms should flow from left to right and top to bottom to avoid frustration and extra time.
4. Consistency - Logical grouping of the information, consistent captioning, use of check-off responses, and table input decrease time needed to complete.
5. Attractiveness - Keep forms attractive. Forms should look uncluttered, appear organized and logical, provide sufficient space for completion, and use different fonts and line weights for capturing attention.

These guidelines for good form design are transferable to screen design, but there are unique qualities of screen displays that must be considered. Two unique design problems for screens are the use of color and the use of icons.

Color is an appealing and proven way to facilitate computer input and allows for contrast of foreground and background. The top five most legible combinations of foreground lettering on background are:

Black on Yellow
Green on White
Blue on White
White on Blue
Yellow on Black

Icons are pictorial, on-screen representations symbolizing computer actions that users may select. In choosing or designing icons, the shapes should be readily recognizable, using standard icons when possible. Icons are useful if meaningful but cuteness should be avoided. Figure 18 shows examples of icons whose meaning may not be readily apparent, and thus should be avoided.



Rabbit - COPY a file

Tree - PRINT

Door - ENTER data

Figure 18: Confusing Icons

6.3 On-Line Input Example

In order to facilitate data collection and reporting, the Software Process Assessment activities in the Data Systems Technology Division at GSFC, in conjunction with the SATC, developed MERIT - Metrics Examination, Reporting and Interpretation Tool. This tool was designed for capturing, maintaining and displaying software metrics data through an on-line process using the World Wide Web. At this time MERIT stores two types of software metrics data: Personnel resources metrics and code metrics but expansion is planned to include error data.

Personnel submit weekly reports of their time spent on a project using the World Wide Web that acts as a front end to MERIT. MERIT then imports the data into the project database. Once the data has been imported into MERIT, the project manager may produce a variety of reports and graphs of the information, allowing them to track progress and discover any problems in the development effort. Administrators (usually not project managers for anonymity of the users) can also use MERIT to check the status of personnel form submissions and notify delinquent users through electronic mail with minimal key strokes.

MERIT is in the initial test phases but reports on it appear promising.

7. Conclusion

Metric programs are initiated to answer a question or provide numerical input to solve a problem. The first step in developing a metrics program is to identify what are the goals or objectives of the program, then stay focused on them. The SATC applies goals to evaluate the quality of products (requirement documents through test applications) and

provide information to project and manage risks. The objectives can be expanded into specific goals using the structure of the Goal/Question/Metric templates.

The second step is to define the attributes that are to be measured. These attributes are generally a subset of the quality attributes and chosen based on the project objectives and goals. If the GQM is used, some of the goals will relate to the attributes.

The next step in developing the metrics program is to clarify and quantify the goals. This is done by specifying questions and identifying metrics and data that is needed. At this point a tool is chosen based on the needs of the project. Some static analysis tools are listed in Appendix A.

The final and a very critical step is to close the loop - provide management with answers to their questions based on the metric analysis. The key to continued success of a metrics program is immediate, visible benefits. It must do the job it was designed to do and supply management with usable information to solve their current problem in a timely fashion.

Appendix A: Code Analysis Tools

Below is a list of code analysis tools identified by the SATC. It is not a comprehensive list. This list does not indicate recommendation by the SATC but could serve as a starting point for tool investigation.

AdaMET, Dynamics Research Corporation, Andover, MA.
Checkpoint, Software Productivity Research, Burlington, MA.
DecisionVision, Software Business Management, Westford, MA.
Logicore Software Development Environment, Logicon, Arlington, VA.
MetKit, Bramer Ltd., Fleet, Hants, UK.
McCabe Object-Oriented Tool, McCabe & Associates, Columbia, MD.
ParaSET, Software Emancipation Technology, Inc., Lexington, MA.
PR:QA, ASTA Incorporated, Nashua, NH.
Rational Environment, Rational, Bethesda, MD.
Spiders-3, Statistica, Inc., Rockville, MD.
UX-Metrics, Set Laboratories Inc., Mulino, OR.

References and Additional Sources of Information

[1] Basili, Victor R., and Rombach, H. Dieter, "The TAME Project: Towards Improvement-Oriented Software Environments", Institute for Advanced Computer Studies, University of Maryland, UMIACS-TR-88-8, January, 1988.

[2] Basili, Victor R., and Rombach, H. Dieter, "Tailoring the Software Process to Project Goals and Environments", Department of Computer Science, University of Maryland, ACM, 1987.

[3] Daskalantonakis, Michael K., "A Practical View of Software Measurement and Implementation Experiences within Motorola", IEEE, 1992.

[4] Fenton, Norman: Software Metrics: A Rigorous Approach, Chapman & Hall, London, UK, 1991.

[5] Grady, Robert, *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall, 1992.

[6] Gillies, Alan, *Software Quality, Theory and Management*, Chapman & Hall Computing, 1992.

[7] Kitchenham, B., Pfleeger, S.L., "Software Quality: The Elusive Target", *IEEE Software*, January, 1996.

[8] Kitchenham, B., Walker, J., "A Quantitative Approach to Monitoring Software Development, *Software Engineering Journal*, January, 1989.

[9] Jones, Capers, *Applied Software Measurement, Assuring Productivity and Quality*, McGraw Hill Inc., 1991.

[10] Moller, K.H., and Paulish, D.J., *Software Metrics, A Practitioner's Guide to Improved Product Development*, Chapman & Hall Computing, 1993.

[11] Putnam, L., Myers, W., *Measures for Excellence: Reliable Software on Time, Within Budget*, Yourdin Press, 1992.

[12] Sommerville, Ian, *Software Engineering*, Addison-Wesley Publishing Company, 1992.

[13] Zuse, Horst: Software Complexity: Measures and Methods, Walter de Gruyter, Berlin, 1990.